Allowing MATLAB to use the Message Passing Interface: MPIMEX

A. Guillen[1], I. Rojas[2], G. Rubio[2], H. Pomares[2], and J. González[2]

[1] Informatics Department, University of Jaen, Spain
Computer Architecture and Computer Technology
[2] Department
University of Granada, Spain

Abstract. The work consists in the development of an new interface that allows MATLAB standalone applications to call MPI standard routines. This interface allows programmers and researchers to design parallel algorithms with the MATLAB application using all its advantages. The new interface is compared with a previous one showing smaller overhead times and an application of the interface over a distributed parallel algorithm is shown.

1 Introduction

MATLAB has binary files to be executed in all the most common platforms: UNIX, Linux, Mac. This program is used by a significant number of researchers and engineers to develop their applications and test models, however, when parallel programming is tuckled, MATLAB does not provide a mechanism to exploit explicit parallelism. More concretely, the Message Passing Interface standard, which is one of the most used libraries in parallel programming, is not supported. In [3] an interface to call MPI [2] functions was developed, however, it is only possible to use it when using Linux Operating System (OS) in a x86 architecture and for the concrete implementation LAM/MPI not considering others like Sun MPI, OpenMPI, etc.

This paper presents a new interface for MATLAB so its applications can invoke MPI functions following the standard and ensuring the possibility of be run in any platform where MATLAB has binaries to be executed on. The applications must be deployed using the MATLAB Compiler so no instances of MATLAB are required to be running at the same time, this is specially adequate for clusters. Thus, the rest of the paper is organized as follows: Section 2 will introduce briefly the MPI standard, then Section 3 will comment the MATLAB Compiler, in Section 4 the new interface will be exposed and in Section 5 a comparative between the new interface and a previous one will be shown as well as an example of a distributed genetic algorithm that was coded in MATLAB using the new interface.

2 Message Passing Interface: MPI

As it is defined in *http://www-unix.mcs.anl.gov/mpi/*, MPI is:

> a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementers, and users.

Among the advantages of, MPI that have made this library well known, are:

- The MPI standard is freely available.
- MPI was designed for high performance on both massively parallel machines and on workstation clusters.
- MPI is widely available, with both free available and vendor-supplied implementations.
- MPI was developed by a broadly based committee of vendors, implementers, and users.
- Information for implementers of MPI is available.
- Test Suites for MPI implementations are available.

The Message Passing Interface was designed in order to provide a programming library for inter-process communication in computer networks, which could be formed by heterogeneous computers. The library is available in many languages such us C, C++, Java, .NET, python, Ocaml, etc.

MPI is the most used library for inter-communication in High-performance computing (HPC) application. There are several vendors and public implementations availables OpenMPI http://www.open-mpi.org/, LAM-MPI http://www.lam-mpi.org/ and MPICH http://www-unix.mcs.anl.gov/mpi/mpich1/, for instance.

3 MATLAB Compiler

MATLAB software has available a tool called *Compiler* which allows MATLAB to generate executable applications (stand-alones) that can be run independently of MATLAB, this is, there is no need of having MATLAB installed in the computer to run the application. The stand-alone requires a set of libraries which can be distributed after being generated with MATLAB, this libraries start the *Component Runtime* (MCR) that interprets the .m files as the MATLAB application would do.

A *Component Technology File* (CTF) is generated during the compilation process. This file contains all the .m files that form the deployed application after being compressed and encrypted so there is no way to access the original source code. When the application is run for the first time, the content of this file is decompressed and a new directory is generated.

The process that MATLAB follows to generate a stand-alone application is made automatically and totally transparent to the user so he only has to specify the .m files that compose the application to be deployed and MATLAB will perform the following operations:

– Dependence analysis between the .m files
– Code generation: the C or C++ code interface is generated in this step.
– File creation: once the dependencies are solved, the .m files are encrypted and compressed in the CTF.
– Compilation: the source code of the interface files is compiled.
– Link: the object code is linked with the required MATLAB libraries.

This whole process is depicted in Figure 1.

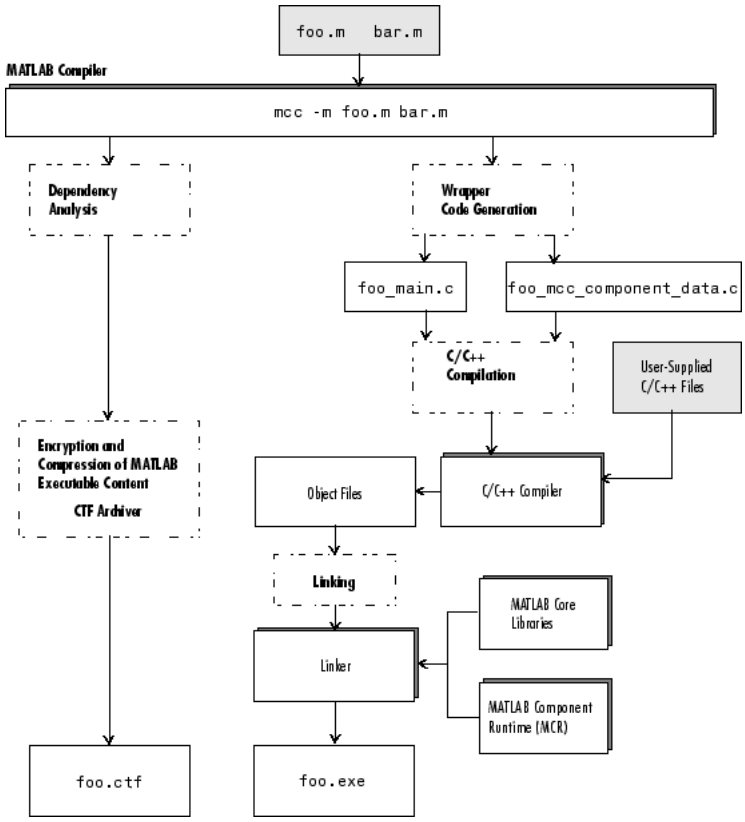


Fig. 1. Deployment process of an application using the MATLAB *Compiler*.

As listed above, there is a code generation step where an interface for the MCR is created. This wrapper files allow a concrete architecture to run the compiled MATLAB code.

3.1 MPIMEX: A new MPI interface for MATLAB

In [3] the *Message Passing Interface ToolBox* (MPITB) was presented. This toolbox has become quite popular, showing the increasing interest of the fussion

between MATLAB and the emerging parallel applications. The main problem that this toolbox has is that it is implemented only for x86 machines running Linux and with the LAM/MPI implementation of MPI. As cited in the Introduction, there are a large variety of implementations of the MPI standard so there is the need of allowing MATLAB use MPI programming in other types of architectures and other MPI implementations. This is the main reason why the new interface proposed in this paper was developed.

MATLAB provides a method to run C/C++ and FORTRAN code within a .m file so the command interpreter can call another function as if it was another .m file. The file that has the .c source code must be written using a special library of functions called mex-functions generating what is know as mex-files [1]. Once the code is written using these special functions, it has to be compiled using the MATLAB *mex* compiler that generates an specific .mexXXX where XXX stands for the concrete architecture MATLAB is running on:

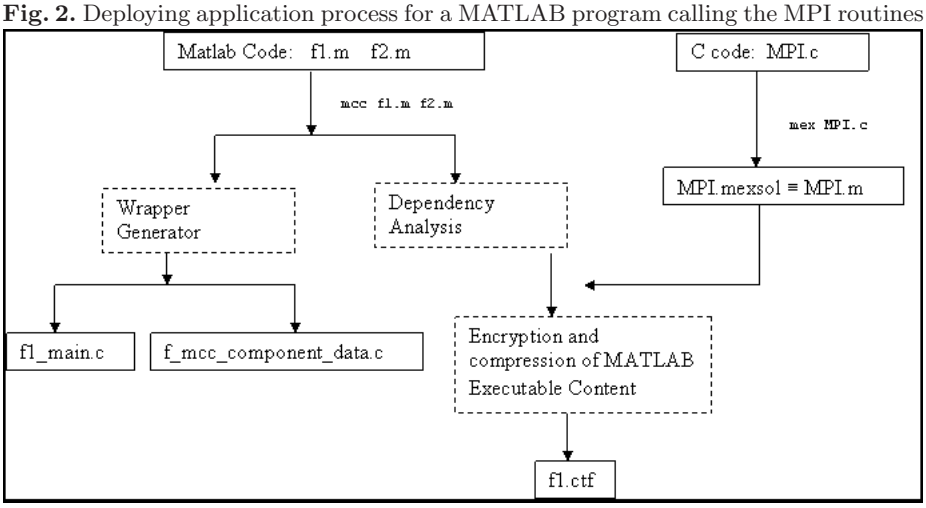
Platform	**MEX extension**
Linux (32-bit)	mexglx
Linux x86-64	mexa64
Macintosh (PPC)	mexmac
Macintosh (Intel)	mexmaci
32-bit Solaris SPARC	mexsol
64-bit Solaris SPARC	mexs64
Windows (32-bit)	mexw32
Windows x64	mexw64

The new interface developed, takes advantage of this feature to invoke the .c standard functions of MPI within the MATLAB source code so when the MATLAB Compiler is executed to deploy the application, those functions are treated as regular .m files. The result is that the deployed application can start the MPI environment and call all the routines defined by the standard. The process to generate a stand-alone application that uses MPI is shown in Figure 2.

Coding in MATLAB The new interface has respected carefully the sintaxis of the standard in order to make easier to use it by people that have already some experience coding with MPI in C although it is still easier than in C because it uses some of the advantages of MATLAB. For example, the initialization of the environment has been simplified comprising three functions of MPI such us MPI_Init, MPI_Comm_size and MPI_Comm_rank so all these parameters can be initialized with a single line of code as an example will show below.

As the interface has been coded in a single file (MPI.mexXXX), a unique function has to be invoked from the MATLAB code, in this function call, there is a parameter that indicates the interface which MPI function will call, so the header of the MPI function is: MPI(MPI_function,...) where MPI_function is a string that has to include the exact name of the C functions excluding the prefix

Fig. 2. Deploying application process for a MATLAB program calling the MPI routines



'MPI_'. For example, to invoke the MPI_Send function which has the following header:

Name: MPI_Send - Performs a standard-mode blocking send.
C Syntax:

#include ¡mpi.h¿
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm

Input Parameters:

buf Initial address of send buffer (choice).
count Number of elements send (nonnegative integer).
datatype Datatype of each send buffer element (handle).
dest Rank of destination (integer).
tag Message tag (integer).
comm Communicator (handle).

the corresponding MPIMEX call within the MATLAB code would be:

MPImex('Send',array, numel(array), 'MPI_DOUBLE', destination, tag, 'MPI_COMM_WORLD');

where all the parameters correspond to the ones defined in the MPI standard although, thanks to MATLAB, there is no need to worry about the type of data of the parameters array, destination and tag.

As commented before, the MPI_Init function has been coded in a slightly different way with the idea of simplifying the coder's task. When MPI('Init') is invoked, it returns the values of the parameters rank and size provided by the functions MPI_Comm_size and MPI_Comm_rank, so the line of code to initialize MPI in MATLAB using MPIMEX is:

[rank,size]=MPImex('Init');

so the value of the return values is performed exactly as if the MPI function was a regular MATLAB function. Although the MPI_Comm_size and MPI_Comm_rank are included in this called, they can be invoked separately using other communicators as MPI allows to define different communicators and it assigns a different rank for the same process. Due to the lack of space, please visit [**?**] for further details on how to use it or contact the authors by e-mail.

4 Experiments

This section shows, in first place, a comparison between the new interface developed and an existing one and then, a real application where MPIMEX has been used.

4.1 Efficiency gain

The portability among the different platforms is not only the advantage over previous toolboxes for message passing in MATLAB but the new interface adds less overhead time when calling MPI routines. To show this efficiency gain, it was compared the new interface with one of the most popular interfaces used so far, MPITB. The two classical message passing routines for the communication between two processes are MPI_Send and MPI_Recv whose header defined by the standard is (the MPI_Send has been described previously):

Name: MPI_Recv - Performs a standard-mode blocking receive.
C Syntax:
#include ¡mpi.h¿
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,int source, int tag, MPI_Comm comm, MPI_S
Input Parameters:
count Maximum number of elements to receive (integer).
datatype Datatype of each receive buffer entry (handle).
source Rank of source (integer).
tag Message tag (integer).
comm Communicator (handle).
Output Parameters
buf Initial address of receive buffer (choice).
status Status object (status).
IERROR Fortran only: Error status (integer).

A simple program that performs a Send and Recv between two processes running in two processors was implemented. The program was executed 10000 times and on each run, the time lapsed during the MPI function calls was measured. Results are shown in Table 1 for MPI_Send and in Table 2 for MPI_Recv, these data have been graphically represented in Figures 3 y 4 respectively.

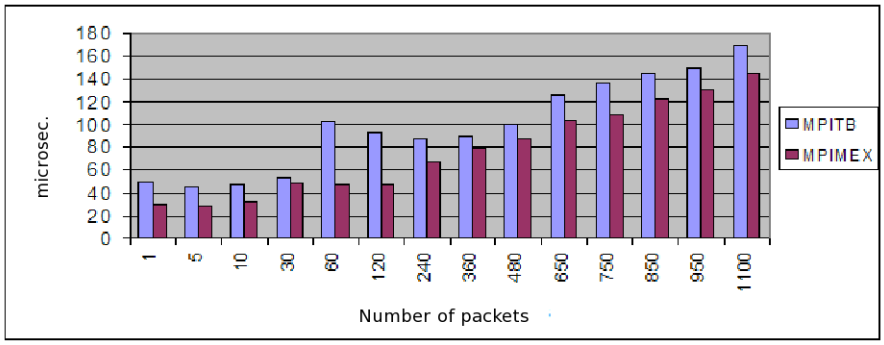


Fig. 3. Comparison between MATLAB MPI interfaces for MPI_Send

Function MPI_Send		
# packets	MPITB	MPIMEX
1	49.85 (168.7)	30.01 (8.2)
5	44.41 (9.8)	28.81 (8.1)
10	47.66 (9.5)	32.33 (10.76)
30	52.69 (7.8)	49.09 (552.1)
60	102.45 (859.7)	46.69 (402.6)
120	93.11 (577.5)	47.52 (281.1)
240	87.92 (409.6)	66.96 (563.1)
360	89.32 (210.3)	79.58 (575.3)
480	100.91 (255.7)	87.32 (417.1)
650	125.09 (461.4)	103.25 (434.2)
750	136.41 (480.1)	109.30 (431.4)
850	145.37 (482.0)	122.20 (444.6)
950	149.08 (326.3)	130.22 (447.8)
1100	169.11 (495.4)	145.61 (449.5)

Table 1. Mean of the time measures in μs. and standard deviation (in brackets) when calling the MPI_Send function using different number of elements.

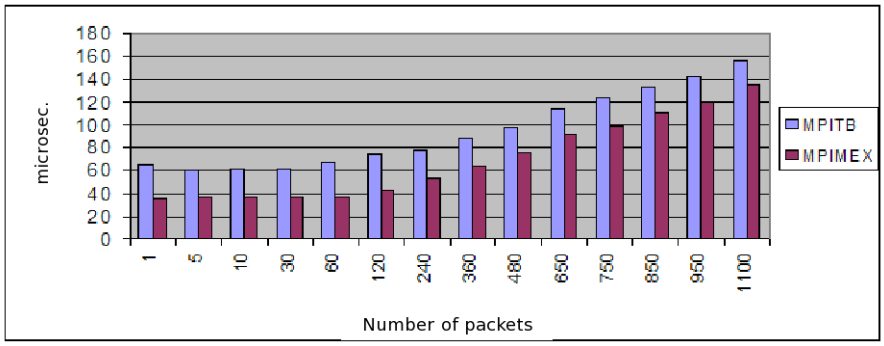


Fig. 4. Comparison between MATLAB MPI interfaces for MPI_Recv

Function MPI_Recv		
# packets	MPITB	MPIMEX
1	65.63 (178.1)	35.91 (17.4)
5	59.92 (20.2)	36.63 (33.8)
10	62 (154.4)	36.28 (33.4)
30	60.74 (9.1)	37.31 (29.5)
60	66.72 (6.6)	36.64 (7.0)
120	73.93 (57.8)	42.56 (10.2)
240	77.61 (11.4)	52.75 (15.1)
360	88.01 (15.7)	63.98 (20.9)
480	97.71 (19.3)	76.12 (24.4)
650	113.97 (24.3)	92.52 (38.2)
750	124.24 (27.9)	99.35 (38.4)
850	133.23 (35.8)	110.89 (45.9)
950	142.53 (44.3)	120.05 (51.3)
1100	156.00 (50.6)	135.43 (65)

Table 2. Mean of the time measures in μs. and standard deviation (in brackets) when calling the MPI_Recv function using different number of elements.

As the results show, there is a larger overhead time when using MPITB than when using the new developed interface. This is the consequence of performing an unique call to a mexfile as explained in the subsection above. As the size of the packet increases, the overhead time becomes inappreciable, however, for fine grained applications where there exists many communications steps, this overhead time can become crucial for the application to be fasted.

4.2 Application over a distributed heterogeneous genetic algorithm

This section shows how this new interface becomes quite useful for model developing, making it fast and simple. The algorithm presented in [6] was able to be executed in a Sun Fire E15K. This machine can reach the number of 106 processors UltraSPARC III Cu 1.2 GHz with a memory of 1/2 TeraByte. The wandwith of the Sun Fire can reach 172.7 Gigabytes per second.

The algorithm consists in a distributed heterogeneous genetic algorithm that has the task of design Radial Basis Function Neural Networks (RBFNN) to approximate functions [7]. The parallelism that can be extrated from this application has two perspectives: data parallelism and functional parallelism. The functional parallelism makes reference to the one that can be obtained when distributing the different tasks so, as was demonstrated in [4, 5] this kind of parallelism, when applied to genetic algorithms does not only increase the efficiency but also improves the results. The data parallelism can be applied to genetic algorithms from two perspectives: referencing the data as the individuals or as the input for the problem. In this case, the first one was considered so an initial population of 300 individuals processed by a initial set of three specialized island was evolved. Then, the population was divided and processed by other groups of island as it is shown in Figure **??**.

The algorithm was executed using a synthetic function to be approximated and the execution times are shown in Table 3 and in Figure 5. The speedup obtained thanks to the parallelism is represented in Figure 6.

	Execution time
3 Proc.	2620(117.21)
6 Proc.	809.5(5.29)
9 Proc.	504.3(21.59)
12 Proc.	346.3(20.55)
15 Proc.	281.6(25.73)
30 Proc.	165.2(21.07)

Table 3. Execution times in seconds and standard deviation (in brackets).

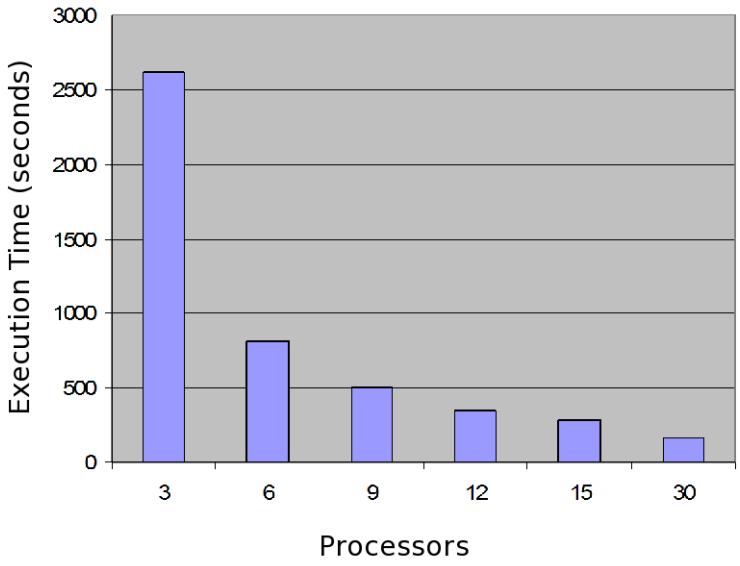


Fig. 5. Execution times in seconds.

5 Conclusions

This paper has presented a new interface that allows MATLAB users to take advantage of the message passing paradigm so they can design parallel applications using the MPI standard. The benefits of this new interface in comparison

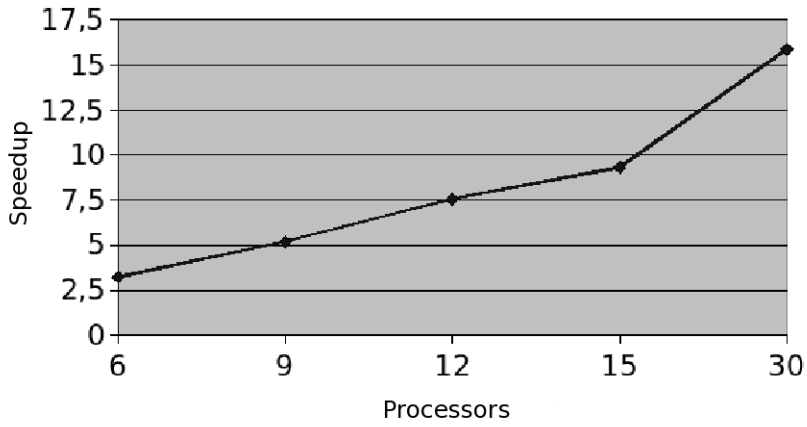


Fig. 6. Speedup obtained when increasing the number of processors.

with previous ones is that it is possible to use it independently of the platform the application will be run on, the implementation of the MPI standard and it also has smaller overhead times.

References

1. http://www.mathworks.com/support/tech-notes/1600/1605.html#intro.
2. http://www-unix.mcs.anl.gov/mpi/, 2005.
3. J. Fernndez, M. Anguita, E. Ros, and J.L. Bernier. SCE Toolboxes for the development of high-level parallel applications. *Lecture Notes in Computer Science*, 3992:518–525, 2006.
4. A. Guillén, I. Rojas, J. Gonz ález, H. Pomares, L.J. Herrera, and B. Paechter. Improving the Performance of Multi-objective Genetic Algorithm for Function Approximation Through Parallel Islands Specialisation. *Lecture Notes in Artificial Intelligence*, 4304:1127–1132, 2006.
5. A. Guillén, I. Rojas, J. Gonz ález, H. Pomares, L.J. Herrera, and B. Paechter. Boosting the performance of a multiobjective algorithm to design RBFNNs through parallelization. *Lecture Notes in Artificial Intelligence*, 2007.
6. A. Guillen, H. Pomares, J. Gonzalez, I. Rojas, L.J. Herrera, and A. Prieto. Parallel Multi-objective Memetic RBFNNs Design and Feature Selection for Function Approximation Problems. *Lecture Notes in Artificial Intelligence*, 4507:341–349, 2007.
7. J. Park and I. Sandberg. Approximation and Radial Basis Function Networks. *Neural Computation*, 5:305–316, 1993.